Kristi Rifenbark
Scotty's Documentation Services, INC
<kristi@buzzcollectivemarketing.com>

Manual Notes

- This version of the ***Toaster User Manual*** is for the 2.5.3 release of the Yocto Project. To be sure you have the latest version of the manual for this release, go to the Yocto Project documentation page and select the manual from that site. Manuals from the site are more up-to-date than manuals derived from the Yocto Project released TAR files.
- If you located this manual through a web search, the version of the manual might not be the one you want (e.g. the search might have returned a manual much older than the Yocto Project version with which you are working). You can see all Yocto Project major releases by visiting the Releases page. If you need a version of this manual for a different Yocto Project release, visit the Yocto Project documentation page and select the manual set by using the "ACTIVE RELEASES DOCUMENTATION" or "DOCUMENTS ARCHIVE" pull-down menus.
- To report any inaccuracies or problems with this manual, send an email to the Yocto Project discussion group at `yocto@yoctoproject.com` or log into the freenode `#yocto` channel.

Revision History	
Revision 1.8	April 2015
Released with the Yocto Project 1.8 Release.	
Revision 2.0	October 2015
Released with the Yocto Project 2.0 Release.	
Revision 2.1	April 2016
Released with the Yocto Project 2.1 Release.	
Revision 2.2	October 2016
Released with the Yocto Project 2.2 Release.	
Revision 2.3	May 2017
Released with the Yocto Project 2.3 Release.	
Revision 2.4	October 2017
Released with the Yocto Project 2.4 Release.	
Revision 2.5	May 2018
Released with the Yocto Project 2.5 Release.	
Revision 2.5.1	September 2018
The initial document released with the Yocto Project 2.5.1 Release.	
Revision 2.5.2	January 2019
The initial document released with the Yocto Project 2.5.2 Release.	
Revision 2.5.3	March 2019
The initial document released with the Yocto Project 2.5.3 Release.	

Table of Contents

Chapter 1. Introduction¶

Table of Contents

Toaster is a web interface to the Yocto Project's OpenEmbedded build system. The interface enables you to configure and run your builds. Information about builds is collected and stored in a database. You can use Toaster to configure and start builds on multiple remote build servers.

1.1. Toaster Features¶

Toaster allows you to configure and run builds, and it provides extensive information about the build process.

- ***Configure and Run Builds:*** You can use the Toaster web interface to configure and start your builds. Builds started using the Toaster web interface are organized into projects. When you create a project, you are asked to select a release, or version of the build system you want to use for the project builds. As shipped, Toaster supports Yocto Project releases 1.8 and beyond. With the Toaster web interface, you can:
  - Browse layers listed in the various layer sources that are available in your project (e.g. the OpenEmbedded Layer Index at http://layers.openembedded.org/layerindex/).
  - Browse images, recipes, and machines provided by those layers.
  - Import your own layers for building.
  - Add and remove layers from your configuration.
  - Set configuration variables.
  - Select a target or multiple targets to build.

- Start your builds.

  Toaster also allows you to configure and run your builds from the command line, and switch between the command line and the web interface at any time. Builds started from the command line appear within a special Toaster project called "Command line builds".

- ***Information About the Build Process:*** Toaster also records extensive information about your builds. Toaster collects data for builds you start from the web interface and from the command line as long as Toaster is running.

  Note
  You must start Toaster before the build or it will not collect build data.

  With Toaster you can:

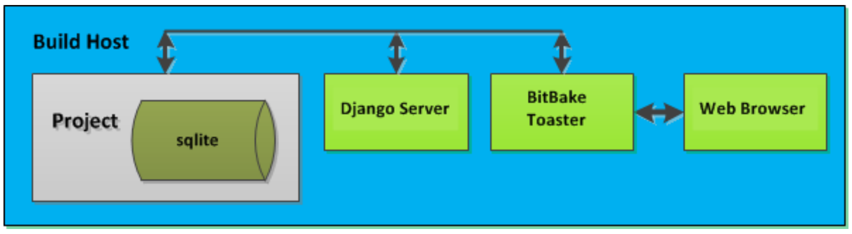
  - See what was built (recipes and packages) and what packages were installed into your final image.
  - Browse the directory structure of your image.
  - See the value of all variables in your build configuration, and which files set each value.
  - Examine error, warning, and trace messages to aid in debugging.
  - See information about the BitBake tasks executed and reused during your build, including those that used shared state.
  - See dependency relationships between recipes, packages, and tasks.
  - See performance information such as build time, task time, CPU usage, and disk I/O.

For an overview of Toaster shipped with the Yocto Project 2.5.3 Release, see the "Toaster - Yocto Project 2.2" video.

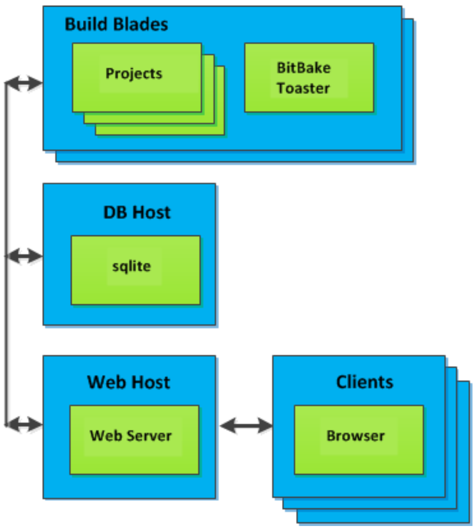
1.2. Installation Options¶

You can set Toaster up to run as a local instance or as a shared hosted service.

When Toaster is set up as a local instance, all the components reside on a single build host. Fundamentally, a local instance of Toaster is suited for a single user developing on a single build host.



Toaster as a hosted service is suited for multiple users developing across several build hosts. When Toaster is set up as a hosted service, its components can be spread across several machines:



Chapter 2. Preparing to Use Toaster¶

Table of Contents

This chapter describes how you need to prepare your system in order to use Toaster.

2.1. Setting Up the Basic System Requirements¶

Before you can use Toaster, you need to first set up your build system to run the Yocto Project. To do this, follow the instructions in the "Preparing the Build Host" section of the Yocto Project Development Tasks Manual. For Ubuntu/Debian, you might also need to do an additional install of pip3.

```
     $ sudo apt-get install python3-pip
```

2.2. Establishing Toaster System Dependencies¶

Toaster requires extra Python dependencies in order to run. A Toaster requirements file named `toaster-requirements.txt` defines the Python dependencies. The requirements file is located in the `bitbake` directory, which is located in the root directory of the Source Directory (e.g. `poky/bitbake/toaster-requirements.txt`). The dependencies appear in a `pip`, install-compatible format.

2.2.1. Install Toaster Packages¶

You need to install the packages that Toaster requires. Use this command:

```
     $ pip3 install --user -r bitbake/toaster-requirements.txt
```

The previous command installs the necessary Toaster modules into a local python 3 cache in your `$HOME` directory. The caches is actually located in `$HOME/.local`. To see what packages have been installed into your `$HOME` directory, do the following:

```
     $ pip3 list installed --local
```

If you need to remove something, the following works:

```
     $ pip3 uninstall PackageNameToUninstall
```

Chapter 3. Setting Up and Using Toaster¶

Table of Contents

3.1. Starting Toaster for Local Development¶

Once you have set up the Yocto Project and installed the Toaster system dependencies as described in the "Preparing to Use Toaster" chapter, you are ready to start Toaster.

Navigate to the root of your Source Directory (e.g. `poky`):

```
     $ cd poky
```

Once in that directory, source the build environment script:

```
     $ source oe-init-build-env
```

Next, from the build directory (e.g. `poky/build`), start Toaster using this command:

```
     $ source toaster start
```

You can now run your builds from the command line, or with Toaster as explained in section "Using the Toaster Web Interface".

To access the Toaster web interface, open your favorite browser and enter the following:

```
http://127.0.0.1:8000
```

3.2. Setting a Different Port¶

By default, Toaster starts on port 8000. You can use the WEBPORT parameter to set a different port. For example, the following command sets the port to "8400":

```
$ source toaster start webport=8400
```

3.3. Setting Up Toaster Without a Web Server¶

You can start a Toaster environment without starting its web server. This is useful for the following:

- Capturing a command-line build's statistics into the Toaster database for examination later.
- Capturing a command-line build's statistics when the Toaster server is already running.
- Having one instance of the Toaster web server track and capture multiple command-line builds, where each build is started in its own "noweb" Toaster environment.

The following commands show how to start a Toaster environment without starting its web server, perform BitBake operations, and then shut down the Toaster environment. Once the build is complete, you can close the Toaster environment. Before closing the environment, however, you should allow a few minutes to ensure the complete transfer of its BitBake build statistics to the Toaster database. If you have a separate Toaster web server instance running, you can watch this command-line build's progress and examine the results as soon as they are posted:

```
$ source toaster start noweb
$ bitbake target
$ source toaster stop
```

3.4. Setting Up Toaster Without a Build Server¶

You can start a Toaster environment with the "New Projects" feature disabled. Doing so is useful for the following:

- Sharing your build results over the web server while blocking others from starting builds on your host.
- Allowing only local command-line builds to be captured into the Toaster database.

Use the following command to set up Toaster without a build server:

```
$ source toaster start nobuild webport=port
```

3.5. Setting up External Access¶

By default, Toaster binds to the loop back address (i.e. localhost), which does not allow access from external hosts. To allow external access, use the WEBPORT parameter to open an address that connects to the network, specifically the IP address that your NIC uses to connect to the network. You can also bind to all IP addresses the computer supports by using the shortcut "0.0.0.0:*port*".

The following example binds to all IP addresses on the host:

```
$ source toaster start webport=0.0.0.0:8400
```

This example binds to a specific IP address on the host's NIC:

```
$ source toaster start webport=192.168.1.1:8400
```

3.6. The Directory for Cloning Layers¶

Toaster creates a _toaster_clones directory inside your Source Directory (i.e. poky) to clone any layers needed for your builds.

Alternatively, if you would like all of your Toaster related files and directories to be in a particular location other than the default, you can set the TOASTER_DIR environment variable, which takes precedence over your current working directory. Setting this environment variable causes Toaster to create and use $TOASTER_DIR./_toaster_clones.

3.7. The Build Directory¶

Toaster creates a build directory within your Source Directory (e.g. poky) to execute the builds.

Alternatively, if you would like all of your Toaster related files and directories to be in a particular location, you can set the `TOASTER_DIR` environment variable, which takes precedence over your current working directory. Setting this environment variable causes Toaster to use `$TOASTER_DIR/build` as the build directory.

3.8. Creating a Django Superuser¶

Toaster is built on the Django framework. Django provides an administration interface you can use to edit Toaster configuration parameters.

To access the Django administration interface, you must create a superuser by following these steps:

1. If you used `pip3`, which is recommended, to set up the Toaster system dependencies, you need be sure the local user path is in your `PATH` list. To append the pip3 local user path, use the following command:

   ```
   $ export PATH=$PATH:$HOME/.local/bin
   ```

2. From the directory containing the Toaster database, which by default is the Build Directory, invoke the `createsuperuser` command from `manage.py`:

   ```
   $ cd ~/poky/build
   $ ../bitbake/lib/toaster/manage.py createsuperuser
   ```

3. Django prompts you for the username, which you need to provide.
4. Django prompts you for an email address, which is optional.
5. Django prompts you for a password, which you must provide.
6. Django prompts you to re-enter your password for verification.

After completing these steps, the following confirmation message appears:

```
Superuser created successfully.
```

Creating a superuser allows you to access the Django administration interface through a browser. The URL for this interface is the same as the URL used for the Toaster instance with "/admin" on the end. For example, if you are running Toaster locally, use the following URL:

```
http://127.0.0.1:8000/admin
```

You can use the Django administration interface to set Toaster configuration parameters such as the build directory, layer sources, default variable values, and BitBake versions.

3.9. Setting Up a Production Instance of Toaster¶

You can use a production instance of Toaster to share the Toaster instance with remote users, multiple users, or both. The production instance is also the setup that can handle heavier loads on the web service. Use the instructions in the following sections to set up Toaster to run builds through the Toaster web interface.

3.9.1. Requirements¶

Be sure you meet the following requirements:

Note
You must comply with all Apache, `mod-wsgi`, and Mysql requirements.

- Have all the build requirements as described in the "Preparing to Use Toaster" chapter.
- Have an Apache webserver.
- Have `mod-wsgi` for the Apache webserver.
- Use the Mysql database server.
- If you are using Ubuntu 16.04, run the following:

  ```
  $ sudo apt-get install apache2 libapache2-mod-wsgi-py3 mysql-server python3-pip libmysqlclient-dev
  ```

- If you are using Fedora 24 or a RedHat distribution, run the following:

  ```
  $ sudo dnf install httpd python3-mod_wsgi python3-pip mariadb-server mariadb-devel python3-devel
  ```

- If you are using openSUSE Leap 42.1, run the following:

  ```
  $ sudo zypper install apache2 apache2-mod_wsgi-python3 python3-pip mariadb mariadb-client python3-devel
  ```

3.9.2. Installation¶

Perform the following steps to install Toaster:

1. Create toaster user and set its home directory to `/var/www/toaster`:

   ```
   $ sudo /usr/sbin/useradd toaster -md /var/www/toaster -s /bin/false
   $ sudo su - toaster -s /bin/bash
   ```

2. Checkout a copy of `poky` into the web server directory. You will be using `/var/www/toaster`:

   ```
   $ git clone git://git.yoctoproject.org/poky
   $ git checkout sumo
   ```

3. Install Toaster dependencies using the --user flag which keeps the Python packages isolated from your system-provided packages:

   ```
   $ cd /var/www/toaster/
   $ pip3 install --user -r ./poky/bitbake/toaster-requirements.txt
   $ pip3 install --user mysqlclient
   ```

   Note
   Isolating these packages is not required but is recommended. Alternatively, you can use your operating system's package manager to install the packages.

4. Configure Toaster by editing `/var/www/toaster/poky/bitbake/lib/toaster/toastermain/settings.py` as follows:

   - Edit the DATABASES settings:

     ```
     DATABASES = {
         'default': {
             'ENGINE': 'django.db.backends.mysql',
             'NAME': 'toaster_data',
             'USER': 'toaster',
             'PASSWORD': 'yourpasswordhere',
             'HOST': 'localhost',
             'PORT': '3306',
        }
     }
     ```

   - Edit the SECRET_KEY:

     ```
     SECRET_KEY = 'your_secret_key'
     ```

   - Edit the STATIC_ROOT:

     ```
     STATIC_ROOT = '/var/www/toaster/static_files/'
     ```

5. Add the database and user to the `mysql` server defined earlier:

   ```
   $ mysql -u root -p
   mysql> CREATE DATABASE toaster_data;
   mysql> CREATE USER 'toaster'@'localhost' identified by 'yourpasswordhere';
   mysql> GRANT all on toaster_data.* to 'toaster'@'localhost';
   mysql> quit
   ```

6. Get Toaster to create the database schema, default data, and gather the statically-served files:

   ```
   $ cd  /var/www/toaster/poky/
   $ ./bitbake/lib/toaster/manage.py migrate
   $ TOASTER_DIR=`pwd` TEMPLATECONF='poky' \
     ./bitbake/lib/toaster/manage.py checksettings
   $ ./bitbake/lib/toaster/manage.py collectstatic
   ```

   In the previous example, from the `poky` directory, the `migrate` command ensures the database schema changes have propagated correctly (i.e. migrations). The next line sets the Toaster root directory `TOASTER_DIR` and the location of the Toaster configuration file `TOASTER_CONF`, which is relative to `TOASTER_DIR`. The `TEMPLATECONF` value reflects the contents of `poky/.templateconf`, and by default, should include the string "poky". For more information on the Toaster configuration file, see the "Configuring Toaster" section.

   This line also runs the `checksettings` command, which configures the location of the Toaster Build Directory. The Toaster root directory `TOASTER_DIR` determines where the Toaster build directory is created on the file system. In the example above, `TOASTER_DIR` is set as follows:

   ```
   /var/www/toaster/poky
   ```

   This setting causes the Toaster build directory to be:

```
/var/www/toaster/poky/build
```

Finally, the `collectstatic` command is a Django framework command that collects all the statically served files into a designated directory to be served up by the Apache web server as defined by `STATIC_ROOT`.

7. Test and/or use the Mysql integration with Toaster's Django web server. At this point, you can start up the normal Toaster Django web server with the Toaster database in Mysql. You can use this web server to confirm that the database migration and data population from the Layer Index is complete.

   To start the default Toaster Django web server with the Toaster database now in Mysql, use the standard start commands:

   ```
   $ source oe-init-build-env
   $ source toaster start
   ```

   Additionally, if Django is sufficient for your requirements, you can use it for your release system and migrate later to Apache as your requirements change.

8. Add an Apache configuration file for Toaster to your Apache web server's configuration directory. If you are using Ubuntu or Debian, put the file here:

   ```
   /etc/apache2/conf-available/toaster.conf
   ```

   If you are using Fedora or RedHat, put it here:

   ```
   /etc/httpd/conf.d/toaster.conf
   ```

   If you are using OpenSUSE, put it here:

   ```
   /etc/apache2/conf.d/toaster.conf
   ```

   Following is a sample Apache configuration for Toaster you can follow:

   ```
   Alias /static /var/www/toaster/static_files
   <Directory /var/www/toaster/static_files>
           <IfModule mod_access_compat.c>
                   Order allow,deny
                   Allow from all
           </IfModule>
           <IfModule !mod_access_compat.c>
                   Require all granted
           </IfModule>
   </Directory>

   <Directory /var/www/toaster/poky/bitbake/lib/toaster/toastermain>
           <Files "wsgi.py">
              Require all granted
           </Files>
   </Directory>

   WSGIDaemonProcess toaster_wsgi python-path=/var/www/toaster/poky/bitbake/lib/toaster:/var/www/toaster/.local/

   WSGIScriptAlias / "/var/www/toaster/poky/bitbake/lib/toaster/toastermain/wsgi.py"
   <Location />
       WSGIProcessGroup toaster_wsgi
   </Location>
   ```

   If you are using Ubuntu or Debian, you will need to enable the config and module for Apache:

   ```
   $ sudo a2enmod wsgi
   $ sudo a2enconf toaster
   $ chmod +x bitbake/lib/toaster/toastermain/wsgi.py
   ```

   Finally, restart Apache to make sure all new configuration is loaded. For Ubuntu, Debian, and OpenSUSE use:

   ```
   $ sudo service apache2 restart
   ```

   For Fedora and RedHat use:

   ```
   $ sudo service httpd restart
   ```

9. Prepare the systemd service to run Toaster builds. Here is a sample configuration file for the service:

   ```
   [Unit]
   Description=Toaster runbuilds

   [Service]
   Type=forking
   User=toaster
   ExecStart=/usr/bin/screen -d -m -S runbuilds /var/www/toaster/poky/bitbake/lib/toaster/runbuilds-service.sh s
   ExecStop=/usr/bin/screen -S runbuilds -X quit
   WorkingDirectory=/var/www/toaster/poky

   [Install]
   WantedBy=multi-user.target
   ```

Prepare the `runbuilds-service.sh` script that you need to place in the `/var/www/toaster/poky/bitbake/lib/toaster/` directory by setting up executable permissions:

```
#!/bin/bash

#export http_proxy=http://proxy.host.com:8080
#export https_proxy=http://proxy.host.com:8080
#export GIT_PROXY_COMMAND=$HOME/bin/gitproxy

cd ~/poky/
source ./oe-init-build-env build
source ../bitbake/bin/toaster $1 noweb
[ "$1" == 'start' ] && /bin/bash
```

10. Run the service:

```
# service runbuilds start
```

Since the service is running in a detached screen session, you can attach to it using this command:

```
$ sudo su - toaster
$ screen -rS runbuilds
```

You can detach from the service again using "Ctrl-a" followed by "d" key combination.

You can now open up a browser and start using Toaster.

3.10. Using the Toaster Web Interface¶

The Toaster web interface allows you to do the following:

- Browse published layers in the OpenEmbedded Layer Index that are available for your selected version of the build system.
- Import your own layers for building.
- Add and remove layers from your configuration.
- Set configuration variables.
- Select a target or multiple targets to build.
- Start your builds.
- See what was built (recipes and packages) and what packages were installed into your final image.
- Browse the directory structure of your image.
- See the value of all variables in your build configuration, and which files set each value.
- Examine error, warning and trace messages to aid in debugging.
- See information about the BitBake tasks executed and reused during your build, including those that used shared state.
- See dependency relationships between recipes, packages and tasks.
- See performance information such as build time, task time, CPU usage, and disk I/O.

3.10.1. Toaster Web Interface Videos¶

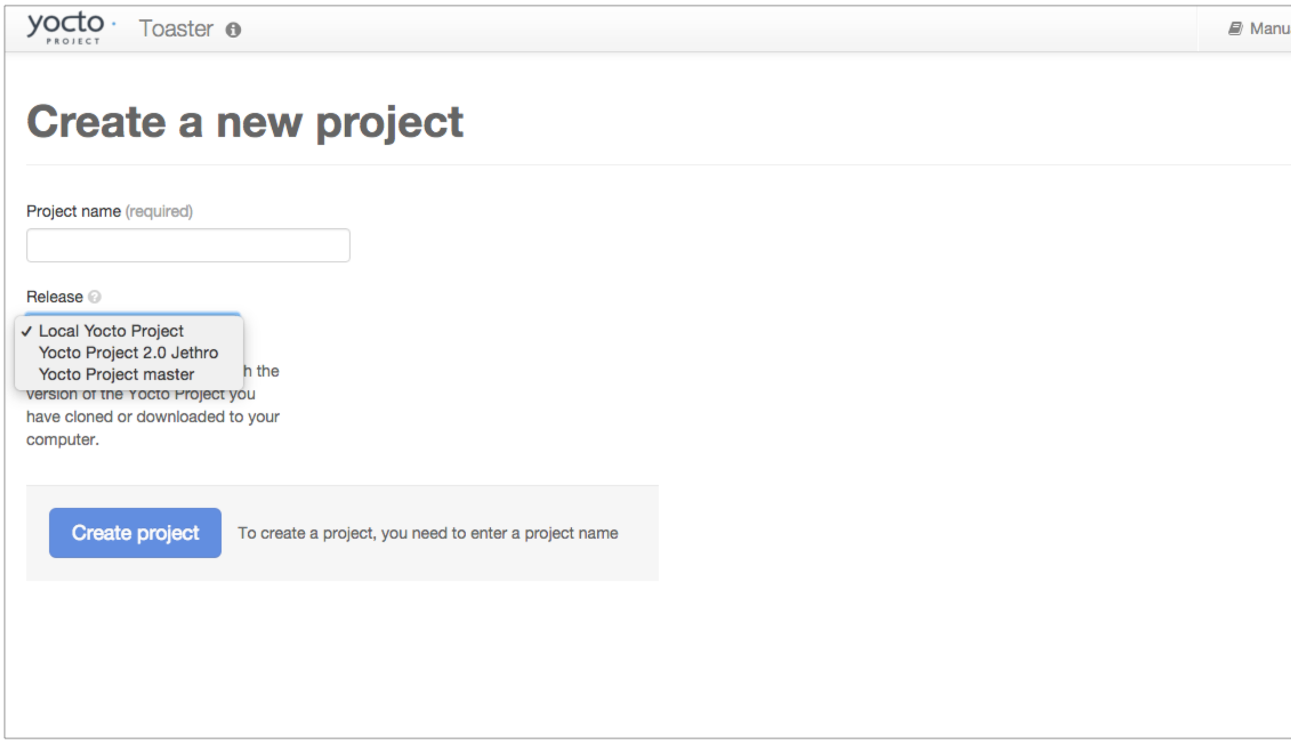
Following are several videos that show how to use the Toaster GUI:

- ***Build Configuration:*** This video overviews and demonstrates build configuration for Toaster.
- ***Build Custom Layers:*** This video shows you how to build custom layers that are used with Toaster.
- ***Toaster Homepage and Table Controls:*** This video goes over the Toaster entry page, and provides an overview of the data manipulation capabilities of Toaster, which include search, sorting and filtering by different criteria.
- ***Build Dashboard:*** This video shows you the build dashboard, a page providing an overview of the information available for a selected build.
- ***Image Information:*** This video walks through the information Toaster provides about images: packages installed and root file system.
- ***Configuration:*** This video provides Toaster build configuration information.
- ***Tasks:*** This video shows the information Toaster provides about the tasks run by the build system.
- ***Recipes and Packages Built:*** This video shows the information Toaster provides about recipes and packages built.
- ***Performance Data:*** This video shows the build performance data provided by Toaster.

3.10.2. Additional Information About the Local Yocto Project Release¶

This section only applies if you have set up Toaster for local development, as explained in the "Starting Toaster for Local Development" section.

When you create a project in Toaster, you will be asked to provide a name and to select a Yocto Project release. One of the release options you will find is called "Local Yocto Project".



Create a new project

Project name (required)

Release

✓ Local Yocto Project
Yocto Project 2.0 Jethro
Yocto Project master

h the version of the Yocto Project you have cloned or downloaded to your computer.

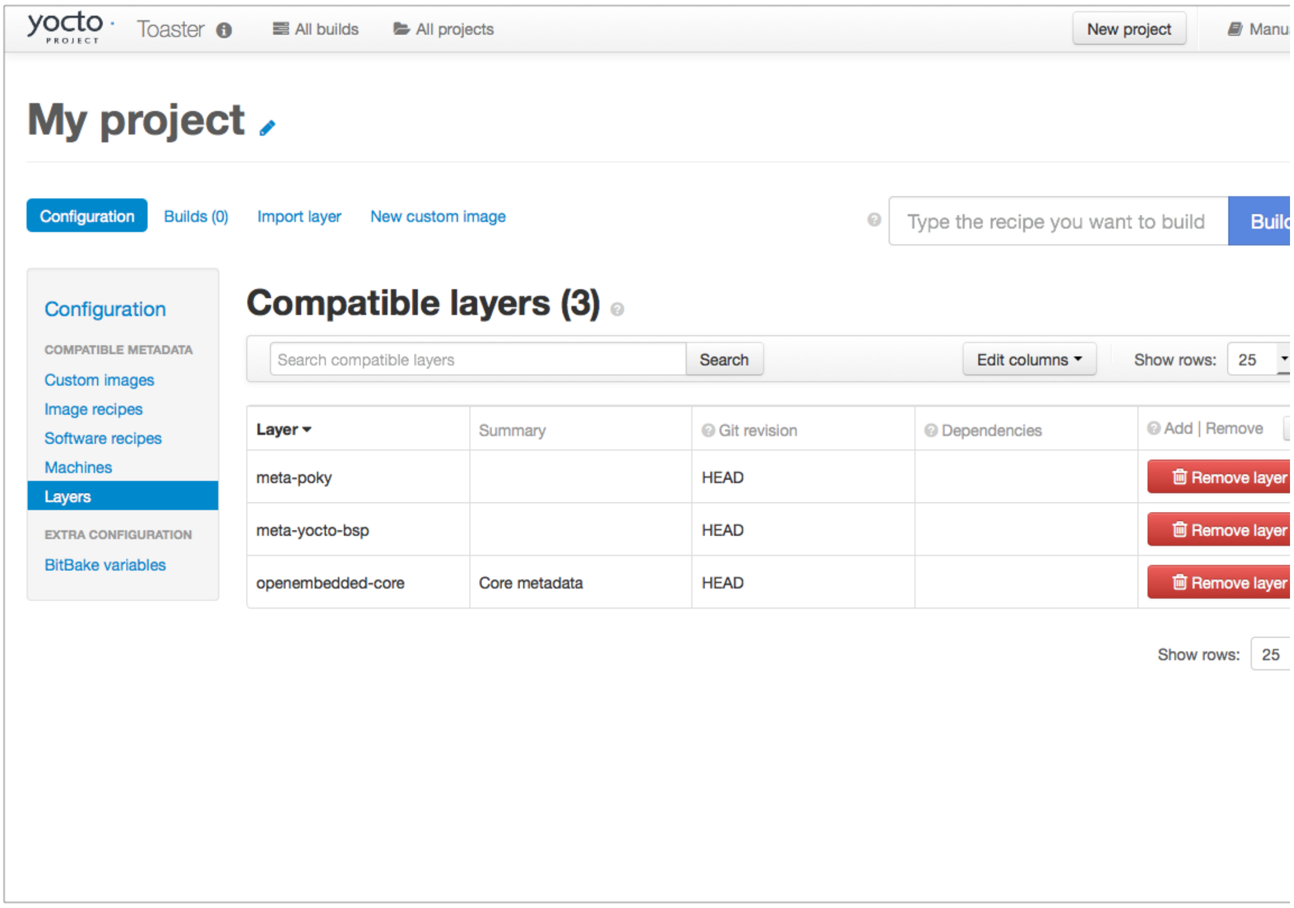
To create a project, you need to enter a project name

When you select the "Local Yocto Project" release, Toaster will run your builds using the local Yocto Project clone you have in your computer: the same clone you are using to run Toaster. Unless you manually update this clone, your builds will always use the same Git revision.

If you select any of the other release options, Toaster will fetch the tip of your selected release from the upstream Yocto Project repository every time you run a build. Fetching this tip effectively means that if your selected release is updated upstream, the Git revision you are using for your builds will change. If you are doing development locally, you might not want this change to happen. In that case, the "Local Yocto Project" release might be the right choice.

However, the "Local Yocto Project" release will not provide you with any compatible layers, other than the three core layers that come with the Yocto Project:

- openembedded-core
- meta-poky
- meta-yocto-bsp

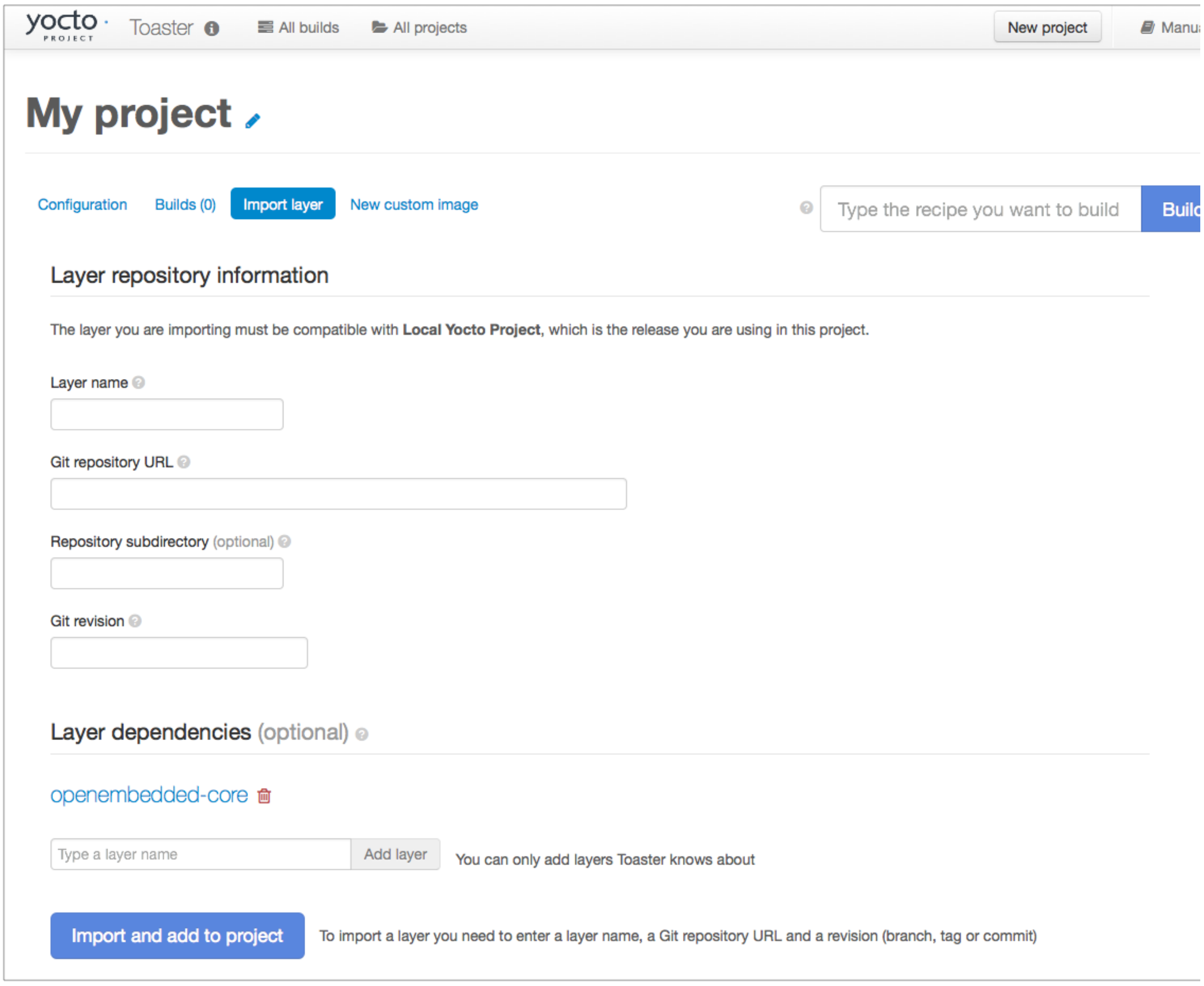


My project

Compatible layers (3)

Layer	Summary	Git revision	Dependencies	Add | Remove
meta-poky		HEAD		Remove layer
meta-yocto-bsp		HEAD		Remove layer
openembedded-core	Core metadata	HEAD		Remove layer

If you want to build any other layers, you will need to manually import them into your Toaster project, using the "Import layer" page.



My project

Layer repository information

The layer you are importing must be compatible with **Local Yocto Project**, which is the release you are using in this project.

Layer name

Git repository URL

Repository subdirectory (optional)

Git revision

Layer dependencies (optional)

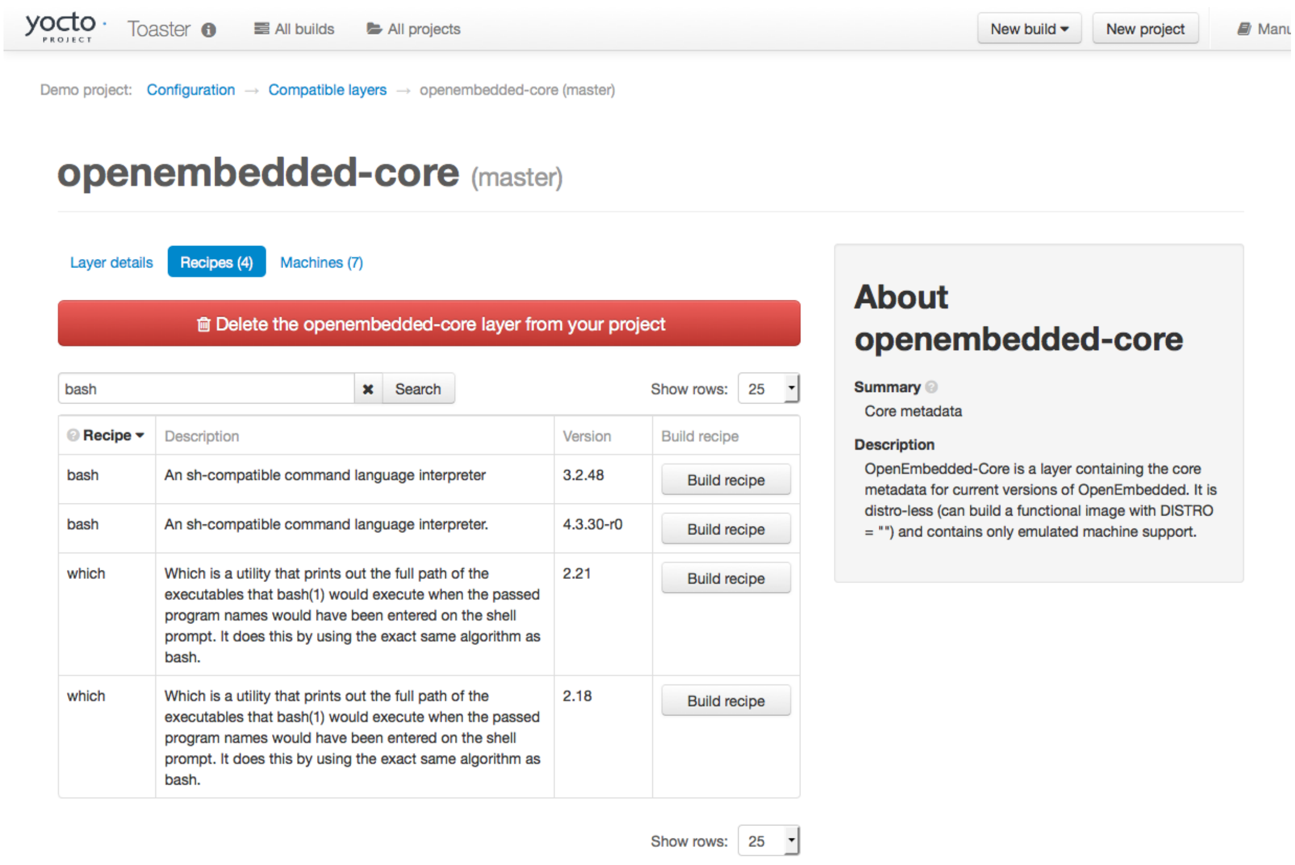
openembedded-core

You can only add layers Toaster knows about

To import a layer you need to enter a layer name, a Git repository URL and a revision (branch, tag or commit)

3.10.3. Building a Specific Recipe Given Multiple Versions¶

Occasionally, a layer might provide more than one version of the same recipe. For example, the `openembedded-core` layer provides two versions of the `bash` recipe (i.e. 3.2.48 and 4.3.30-r0) and two versions of the `which` recipe (i.e. 2.21 and 2.18). The following figure shows this exact scenario:


yocto PROJECT
Toaster
All builds
All projects
New build
New project
Manu
Demo project: Configuration → Compatible layers → openembedded-core (master)
Layer details
Recipes (4)
Machines (7)
Delete the openembedded-core layer from your project
bash
Search
Show rows: 25

openembedded-core (master)

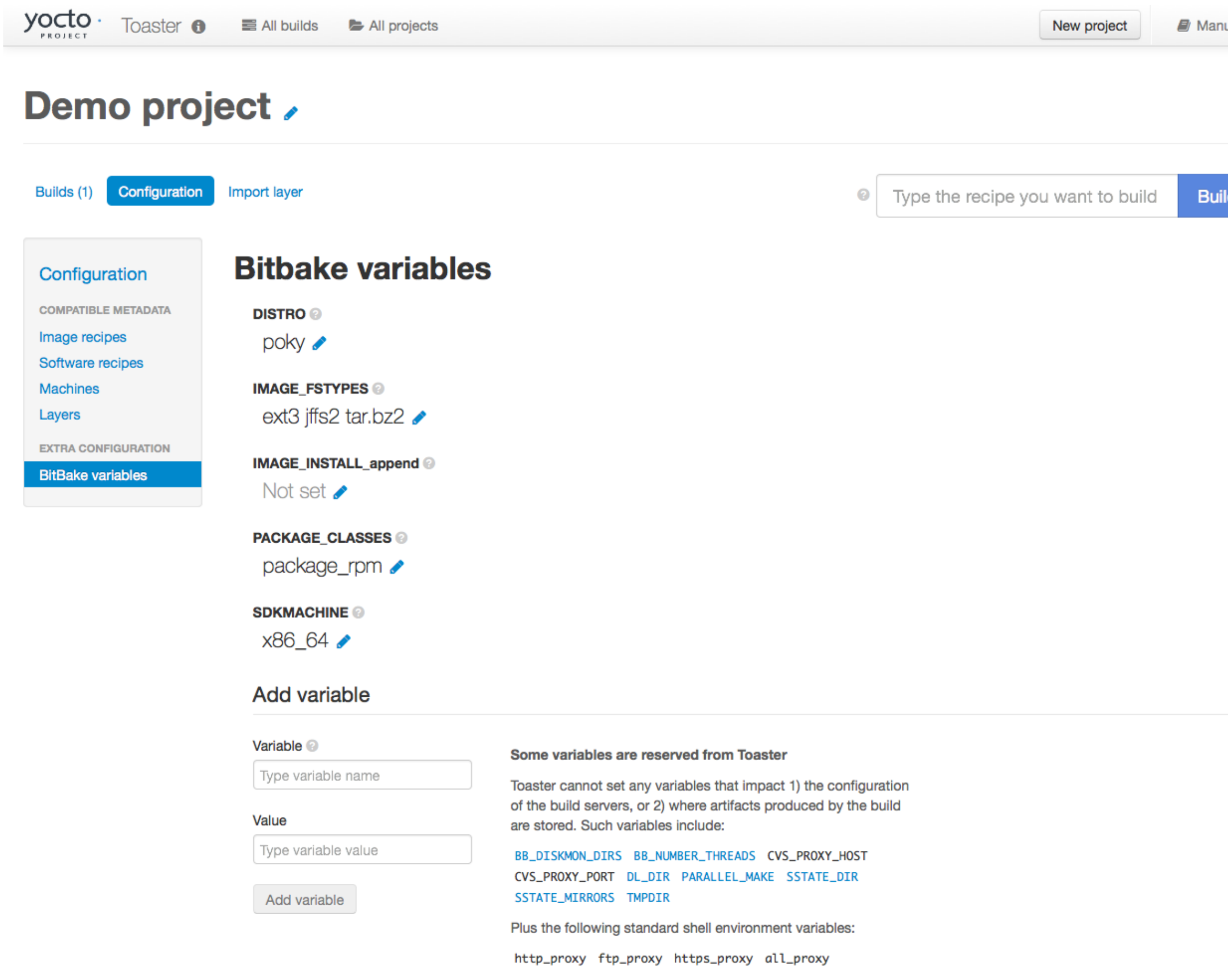
Recipe	Description	Version	Build recipe
bash	An sh-compatible command language interpreter	3.2.48	Build recipe
bash	An sh-compatible command language interpreter.	4.3.30-r0	Build recipe
which	Which is a utility that prints out the full path of the executables that bash(1) would execute when the passed program names would have been entered on the shell prompt. It does this by using the exact same algorithm as bash.	2.21	Build recipe
which	Which is a utility that prints out the full path of the executables that bash(1) would execute when the passed program names would have been entered on the shell prompt. It does this by using the exact same algorithm as bash.	2.18	Build recipe

About openembedded-core

Summary
Core metadata

Description
OpenEmbedded-Core is a layer containing the core metadata for current versions of OpenEmbedded. It is distro-less (can build a functional image with DISTRO = "") and contains only emulated machine support.

By default, the OpenEmbedded build system builds one of the two recipes. For the `bash` case, version 4.3.30-r0 is built by default. Unfortunately, Toaster as it exists, is not able to override the default recipe version. If you would like to build bash 3.2.48, you need to set the `PREFERRED_VERSION` variable. You can do so from Toaster, using the "Add variable" form, which is available in the "BitBake variables" page of the project configuration section as shown in the following screen:



Demo project

Bitbake variables

DISTRO
poky

IMAGE_FSTYPES
ext3 jffs2 tar.bz2

IMAGE_INSTALL_append
Not set

PACKAGE_CLASSES
package_rpm

SDKMACHINE
x86_64

Add variable

Variable
Type variable name

Value
Type variable value

Add variable

Some variables are reserved from Toaster

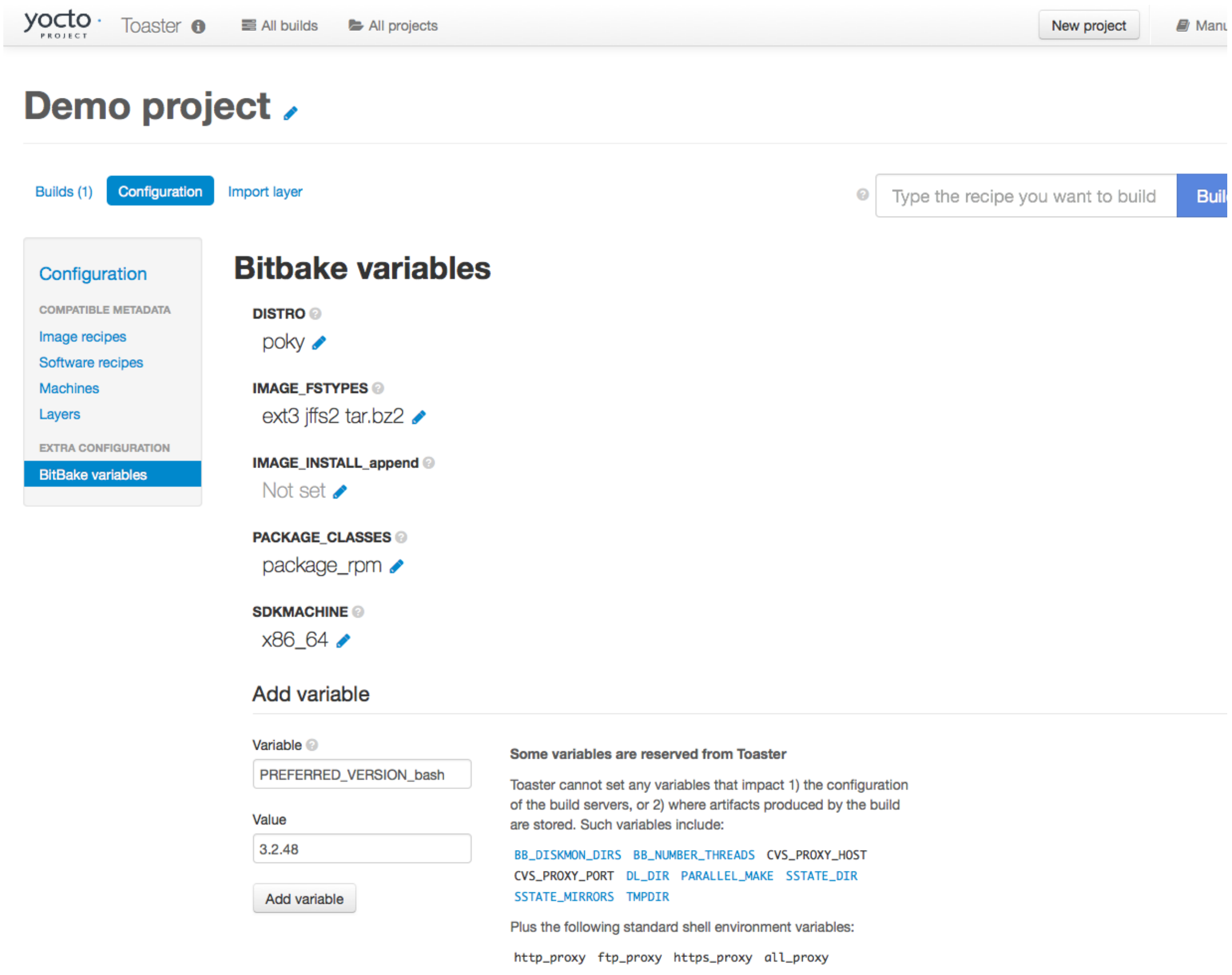
Toaster cannot set any variables that impact 1) the configuration of the build servers, or 2) where artifacts produced by the build are stored. Such variables include:

`BB_DISKMON_DIRS` `BB_NUMBER_THREADS` `CVS_PROXY_HOST` `CVS_PROXY_PORT` `DL_DIR` `PARALLEL_MAKE` `SSTATE_DIR` `SSTATE_MIRRORS` `TMPDIR`

Plus the following standard shell environment variables:

`http_proxy` `ftp_proxy` `https_proxy` `all_proxy`

To specify `bash` 3.2.48 as the version to build, enter "PREFERRED_VERSION_bash" in the "Variable" field, and "3.2.48" in the "Value" field. Next, click the "Add variable" button:



Demo project

Bitbake variables

DISTRO
poky

IMAGE_FSTYPES
ext3 jffs2 tar.bz2

IMAGE_INSTALL_append
Not set

PACKAGE_CLASSES
package_rpm

SDKMACHINE
x86_64

Add variable

Variable
PREFERRED_VERSION_bash

Value
3.2.48

Some variables are reserved from Toaster

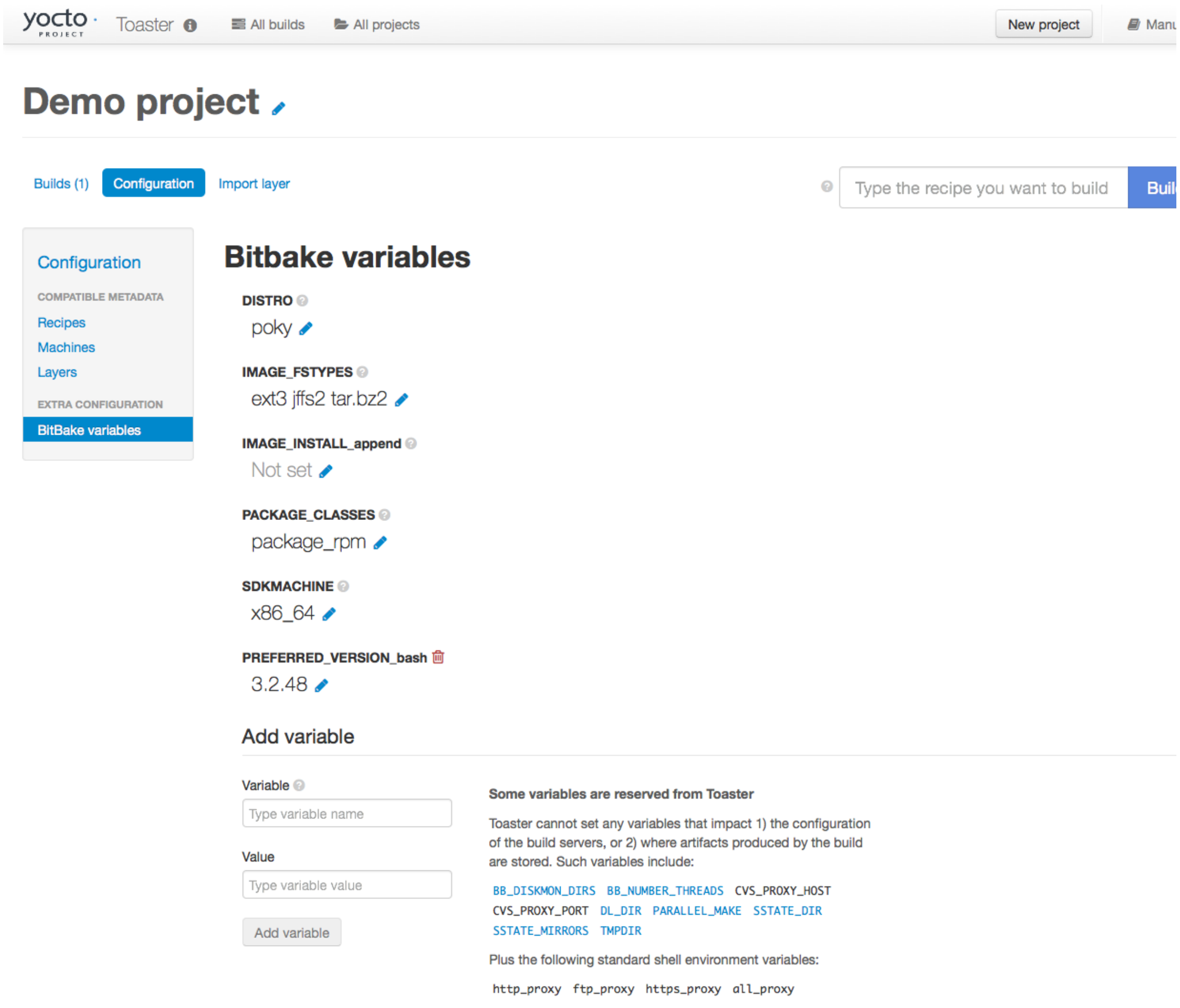
Toaster cannot set any variables that impact 1) the configuration of the build servers, or 2) where artifacts produced by the build are stored. Such variables include:

`BB_DISKMON_DIRS` `BB_NUMBER_THREADS` `CVS_PROXY_HOST` `CVS_PROXY_PORT` `DL_DIR` `PARALLEL_MAKE` `SSTATE_DIR` `SSTATE_MIRRORS` `TMPDIR`

Plus the following standard shell environment variables:

`http_proxy` `ftp_proxy` `https_proxy` `all_proxy`

After clicking the "Add variable" button, the settings for `PREFERRED_VERSION` are added to the bottom of the BitBake variables list. With these settings, the OpenEmbedded build system builds the desired version of the recipe rather than the default version:



Demo project

Bitbake variables

DISTRO
poky

IMAGE_FSTYPES
ext3 jffs2 tar.bz2

IMAGE_INSTALL_append
Not set

PACKAGE_CLASSES
package_rpm

SDKMACHINE
x86_64

PREFERRED_VERSION_bash
3.2.48

Add variable

Variable
Type variable name

Value
Type variable value

Some variables are reserved from Toaster

Toaster cannot set any variables that impact 1) the configuration of the build servers, or 2) where artifacts produced by the build are stored. Such variables include:

`BB_DISKMON_DIRS` `BB_NUMBER_THREADS` `CVS_PROXY_HOST` `CVS_PROXY_PORT` `DL_DIR` `PARALLEL_MAKE` `SSTATE_DIR` `SSTATE_MIRRORS` `TMPDIR`

Plus the following standard shell environment variables:

`http_proxy` `ftp_proxy` `https_proxy` `all_proxy`

Chapter 4. Concepts and Reference¶

Table of Contents

In order to configure and use Toaster, you should understand some concepts and have some basic command reference material available. This final chapter provides conceptual information on layer sources, releases, and JSON configuration files. Also provided is a quick look at some useful `manage.py` commands that are Toaster-specific. Information on `manage.py` commands does exist across the Web and the information in this manual by no means attempts to provide a command comprehensive reference.

4.1. Layer Source¶

In general, a "layer source" is a source of information about existing layers. In particular, we are concerned with layers that you can use with the Yocto Project and Toaster. This chapter describes a particular type of layer source called a "layer index."

A layer index is a web application that contains information about a set of custom layers. A good example of an existing layer index is the OpenEmbedded Layer Index. A public instance of this layer index exists at http://layers.openembedded.org. You can find the code for this layer index's web application at http://git.yoctoproject.org/cgit/cgit.cgi/layerindex-web/.

When you tie a layer source into Toaster, it can query the layer source through a REST API, store the information about the layers in the Toaster database, and then show the information to users. Users are then able to view that information and build layers from Toaster itself without worrying about cloning or editing the BitBake layers configuration file `bblayers.conf`.

Tying a layer source into Toaster is convenient when you have many custom layers that need to be built on a regular basis by a community of developers. In fact, Toaster comes pre-configured with the OpenEmbedded Metadata Index.

Note

You do not have to use a layer source to use Toaster. Tying into a layer source is optional.

4.1.1. Setting Up and Using a Layer Source¶

To use your own layer source, you need to set up the layer source and then tie it into Toaster. This section describes how to tie into a layer index in a manner similar to the way Toaster ties into the OpenEmbedded Metadata Index.

4.1.1.1. Understanding Your Layers¶

The obvious first step for using a layer index is to have several custom layers that developers build and access using the Yocto Project on a regular basis. This set of layers needs to exist and you need to be familiar with where they reside. You will need that information when you set up the code for the web application that "hooks" into your set of layers.

For general information on layers, see the "The Yocto Project Layer Model" section in the Yocto Project Overview and Concepts Manual. For information on how to create layers, see the "Understanding and Creating Layers" section in the Yocto Project Development Tasks Manual.

4.1.1.2. Configuring Toaster to Hook Into Your Layer Index¶

If you want Toaster to use your layer index, you must host the web application in a server to which Toaster can connect. You also need to give Toaster the information about your layer index. In other words, you have to configure Toaster to use your layer index. This section describes two methods by which you can configure and use your layer index.

In the previous section, the code for the OpenEmbedded Metadata Index (i.e. http://layers.openembedded.org) was referenced. You can use this code, which is at http://git.yoctoproject.org/cgit/cgit.cgi/layerindex-web/, as a base to create your own layer index.

4.1.1.2.1. Use the Administration Interface¶

Access the administration interface through a browser by entering the URL of your Toaster instance and adding "`/admin`" to the end of the URL. As an example, if you are running Toaster locally, use the following URL:

```
http://127.0.0.1:8000/admin
```

The administration interface has a "Layer sources" section that includes an "Add layer source" button. Click that button and provide the required information. Make sure you select "layerindex" as the layer source type.

4.1.1.2.2. Use the Fixture Feature¶

The Django fixture feature overrides the default layer server when you use it to specify a custom URL. To use the fixture feature, create (or edit) the file `bitbake/lib/toaster.orm/fixtures/custom.xml`, and then set the following Toaster setting to your custom URL:

```
<?xml version="1.0" ?>
<django-objects version="1.0">
  <object model="orm.toastersetting" pk="100">
              <field name="name" type="CharField">CUSTOM_LAYERINDEX_SERVER</field>
              <field name="value" type="CharField">https://layers.my_organization.org/layerindex/branch/master/l
  </object>
<django-objects>
```

When you start Toaster for the first time, or if you delete the file `toaster.sqlite` and restart, the database will populate cleanly from this layer index server.

Once the information has been updated, verify the new layer information is available by using the Toaster web interface. To do that, visit the "All compatible layers" page inside a Toaster project. The layers from your layer source should be listed there.

If you change the information in your layer index server, refresh the Toaster database by running the following command:

```
$ bitbake/lib/toaster/manage.py lsupdates
```

If Toaster can reach the API URL, you should see a message telling you that Toaster is updating the layer source information.

4.2. Releases¶

When you create a Toaster project using the web interface, you are asked to choose a "Release." In the context of Toaster, the term "Release" refers to a set of layers and a BitBake version the OpenEmbedded build system uses to build something. As shipped, Toaster is pre-configured with releases that correspond to Yocto Project release branches. However, you can modify, delete, and create new releases according to your needs. This section provides some background information on releases.

4.2.1. Pre-Configured Releases¶

As shipped, Toaster is configured to use a specific set of releases. Of course, you can always configure Toaster to use any release. For example, you might want your project to build against a specific commit of any of the "out-of-the-box" releases. Or, you might want your project to build against different revisions of OpenEmbedded and BitBake.

As shipped, Toaster is configured to work with the following releases:

- ***Yocto Project 2.5.3 "Sumo" or OpenEmbedded "Sumo":*** This release causes your Toaster projects to build against the head of the sumo branch at http://git.yoctoproject.org/cgit/cgit.cgi/poky/log/?h=rocko or http://git.openembedded.org/openembedded-core/commit/?h=rocko.
- ***Yocto Project "Master" or OpenEmbedded "Master":*** This release causes your Toaster Projects to build against the head of the master branch, which is where active development takes place, at http://git.yoctoproject.org/cgit/cgit.cgi/poky/log/ or http://git.openembedded.org/openembedded-core/log/.
- ***Local Yocto Project or Local OpenEmbedded:*** This release causes your Toaster Projects to build against the head of the `poky` or `openembedded-core` clone you have local to the machine running Toaster.

4.3. Configuring Toaster¶

In order to use Toaster, you must configure the database with the default content. The following subsections describe various aspects of Toaster configuration.

4.3.1. Configuring the Workflow¶

The `bldcontrol/management/commands/checksettings.py` file controls workflow configuration. The following steps outline the process to initially populate this database.

1. The default project settings are set from `orm/fixtures/settings.xml`.
2. The default project distro and layers are added from `orm/fixtures/poky.xml` if poky is installed. If poky is not installed, they are added from `orm/fixtures/oe-core.xml`.
3. If the `orm/fixtures/custom.xml` file exists, then its values are added.
4. The layer index is then scanned and added to the database.

Once these steps complete, Toaster is set up and ready to use.

4.3.2. Customizing Pre-Set Data¶

The pre-set data for Toaster is easily customizable. You can create the `orm/fixtures/custom.xml` file to customize the values that go into to the database. Customization is additive, and can either extend or completely replace the existing values.

You use the `orm/fixtures/custom.xml` file to change the default project settings for the machine, distro, file images, and layers. When creating a new project, you can use the file to define the offered alternate project release selections. For example, you can add one or more additional selections that present custom layer sets or distros, and any other local or proprietary content.

Additionally, you can completely disable the content from the `oe-core.xml` and `poky.xml` files by defining the section shown below in the `settings.xml` file. For example, this option is particularly useful if your custom configuration defines fewer releases or layers than the default fixture files.

The following example sets "name" to "CUSTOM_XML_ONLY" and its value to "True".

```
<object model="orm.toastersetting" pk="99">
  <field type="CharField" name="name">CUSTOM_XML_ONLY</field>
  <field type="CharField" name="value">True</field>
</object>
```

4.3.3. Understanding Fixture File Format¶

The following is an overview of the file format used by the `oe-core.xml`, `poky.xml`, and `custom.xml` files.

The following subsections describe each of the sections in the fixture files, and outline an example section of the XML code. you can use to help understand this information and create a local `custom.xml` file.

4.3.3.1. Defining the Default Distro and Other Values¶

This section defines the default distro value for new projects. By default, it reserves the first Toaster Setting record "1". The following demonstrates how to set the project default value for DISTRO:

```
<!-- Set the project default value for DISTRO -->
<object model="orm.toastersetting" pk="1">
  <field type="CharField" name="name">DEFCONF_DISTRO</field>
  <field type="CharField" name="value">poky</field>
</object>
```

You can override other default project values by adding additional Toaster Setting sections such as any of the settings coming from the `settings.xml` file. Also, you can add custom values that are included in the BitBake environment. The "pk" values must be unique. By convention, values that set default project values have a "DEFCONF" prefix.

4.3.3.2. Defining BitBake Version¶

The following defines which version of BitBake is used for the following release selection:

```
<!-- Bitbake versions which correspond to the metadata release -->
<object model="orm.bitbakeversion" pk="1">
  <field type="CharField" name="name">rocko</field>
  <field type="CharField" name="giturl">git://git.yoctoproject.org/poky</field>
  <field type="CharField" name="branch">rocko</field>
  <field type="CharField" name="dirpath">bitbake</field>
</object>
```

4.3.3.3. Defining Release¶

The following defines the releases when you create a new project.

```
<!-- Releases available -->
<object model="orm.release" pk="1">
  <field type="CharField" name="name">rocko</field>
  <field type="CharField" name="description">Yocto Project 2.4 "Rocko"</field>
  <field rel="ManyToOneRel" to="orm.bitbakeversion" name="bitbake_version">1</field>
  <field type="CharField" name="branch_name">rocko</field>
  <field type="TextField" name="helptext">Toaster will run your builds using the tip of the <a href="http://git.yc
</object>
```

The "pk" value must match the above respective BitBake version record.

4.3.3.4. Defining the Release Default Layer Names¶

The following defines the default layers for each release:

```
<!-- Default project layers for each release -->
<object model="orm.releasedefaultlayer" pk="1">
  <field rel="ManyToOneRel" to="orm.release" name="release">1</field>
  <field type="CharField" name="layer_name">openembedded-core</field>
</object>
```

The 'pk' values in the example above should start at "1" and increment uniquely. You can use the same layer name in multiple releases.

4.3.3.5. Defining Layer Definitions¶

Layer definitions are the most complex. The following defines each of the layers, and then defines the exact layer version of the layer used for each respective release. You must have one `orm.layer` entry for each layer. Then, with each entry you need a set of `orm.layer_version` entries that connects the layer with each release that includes the layer. In general all releases include the layer.

```
<object model="orm.layer" pk="1">
  <field type="CharField" name="name">openembedded-core</field>
  <field type="CharField" name="layer_index_url"></field>
  <field type="CharField" name="vcs_url">git://git.yoctoproject.org/poky</field>
  <field type="CharField" name="vcs_web_url">http://git.yoctoproject.org/cgit/cgit.cgi/poky</field>
  <field type="CharField" name="vcs_web_tree_base_url">http://git.yoctoproject.org/cgit/cgit.cgi/poky/tree/%path%?
  <field type="CharField" name="vcs_web_file_base_url">http://git.yoctoproject.org/cgit/cgit.cgi/poky/tree/%path%?
</object>
<object model="orm.layer_version" pk="1">
  <field rel="ManyToOneRel" to="orm.layer" name="layer">1</field>
  <field type="IntegerField" name="layer_source">0</field>
  <field rel="ManyToOneRel" to="orm.release" name="release">1</field>
  <field type="CharField" name="branch">rocko</field>
  <field type="CharField" name="dirpath">meta</field>
</object>
<object model="orm.layer_version" pk="2">
  <field rel="ManyToOneRel" to="orm.layer" name="layer">1</field>
  <field type="IntegerField" name="layer_source">0</field>
  <field rel="ManyToOneRel" to="orm.release" name="release">2</field>
  <field type="CharField" name="branch">HEAD</field>
  <field type="CharField" name="commit">HEAD</field>
  <field type="CharField" name="dirpath">meta</field>
</object>
<object model="orm.layer_version" pk="3">
  <field rel="ManyToOneRel" to="orm.layer" name="layer">1</field>
  <field type="IntegerField" name="layer_source">0</field>
  <field rel="ManyToOneRel" to="orm.release" name="release">3</field>
  <field type="CharField" name="branch">master</field>
  <field type="CharField" name="dirpath">meta</field>
</object>
```

The layer "pk" values above must be unique, and typically start at "1". The layer version "pk" values must also be unique across all layers, and typically start at "1".

4.4. Remote Toaster Monitoring¶

Toaster has an API that allows remote management applications to directly query the state of the Toaster server and its builds in a machine-to-machine manner. This API uses the REST interface and the transfer of JSON files. For example, you might monitor a build inside a container through well supported known HTTP ports in order to easily access a Toaster server inside the container. In this example, when you use this direct JSON API, you avoid having web page parsing against the display the user sees.

4.4.1. Checking Health¶

Before you use remote Toaster monitoring, you should do a health check. To do this, ping the Toaster server using the following call to see if it is still alive:

```
http://host:port/health
```

Be sure to provide values for *host* and *port*. If the server is alive, you will get the response HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head><title>Toaster Health</title></head>
  <body>Ok</body>
</html>
```

4.4.2. Determining Status of Builds in Progress¶

Sometimes it is useful to determine the status of a build in progress. To get the status of pending builds, use the following call:

```
http://host:port/toastergui/api/building
```

Be sure to provide values for *host* and *port*. The output is a JSON file that itemizes all builds in progress. This file includes the time in seconds since each respective build started as well as the progress of the cloning, parsing, and task execution. The following is sample output for a build in progress:

```
{"count": 1,
 "building": [
   {"machine": "beaglebone",
      "seconds": "463.869",
      "task": "927:2384",
      "distro": "poky",
      "clone": "1:1",
      "id": 2,
      "start": "2017-09-22T09:31:44.887Z",
      "name": "20170922093200",
      "parse": "818:818",
      "project": "my_rocko",
      "target": "core-image-minimal"
      }]
}
```

The JSON data for this query is returned in a single line. In the previous example the line has been artificially split for readability.

4.4.3. Checking Status of Builds Completed¶

Once a build is completed, you get the status when you use the following call:

```
http://host:port/toastergui/api/builds
```

Be sure to provide values for *host* and *port*. The output is a JSON file that itemizes all complete builds, and includes build summary information. The following is sample output for a completed build:

```
{"count": 1,
 "builds": [
   {"distro": "poky",
      "errors": 0,
      "machine":
      "beaglebone",
      "project": "my_rocko",
      "stop": "2017-09-22T09:26:36.017Z",
      "target": "quilt-native",
      "seconds": "78.193",
      "outcome": "Succeeded",
      "id": 1,
      "start": "2017-09-22T09:25:17.824Z",
      "warnings": 1,
      "name": "20170922092618"
      }]
}
```

The JSON data for this query is returned in a single line. In the previous example the line has been artificially split for readability.

4.4.4. Determining Status of a Specific Build¶

Sometimes it is useful to determine the status of a specific build. To get the status of a specific build, use the following call:

```
http://host:port/toastergui/api/build/ID
```

Be sure to provide values for *host*, *port*, and *ID*. You can find the value for *ID* from the Builds Completed query. See the "Checking Status of Builds Completed" section for more information.

The output is a JSON file that itemizes the specific build and includes build summary information. The following is sample output for a specific build:

```
{"build":
  {"distro": "poky",
   "errors": 0,
   "machine": "beaglebone",
   "project": "my_rocko",
   "stop": "2017-09-22T09:26:36.017Z",
   "target": "quilt-native",
   "seconds": "78.193",
   "outcome": "Succeeded",
   "id": 1,
   "start": "2017-09-22T09:25:17.824Z",
   "warnings": 1,
   "name": "20170922092618",
   "cooker_log": "/opt/user/poky/build-toaster-2/tmp/log/cooker/beaglebone/build_20170922_022607.991.log"
   }
}
```

The JSON data for this query is returned in a single line. In the previous example the line has been artificially split for readability.

4.5. Useful Commands¶

In addition to the web user interface and the scripts that start and stop Toaster, command-line commands exist through the `manage.py` management script. You can find general documentation on `manage.py` at the Django site. However, several `manage.py` commands have been created that are specific to Toaster and are used to control configuration and back-end tasks. You can locate these commands in the Source Directory (e.g. `poky`) at `bitbake/lib/manage.py`. This section documents those commands.

Notes

- When using `manage.py` commands given a default configuration, you must be sure that your working directory is set to the Build Directory. Using `manage.py` commands from the Build Directory allows Toaster to find the `toaster.sqlite` file, which is located in the Build Directory.
- For non-default database configurations, it is possible that you can use `manage.py` commands from a directory other than the Build Directory. To do so, the `toastermain/settings.py` file must be configured to point to the correct database backend.

4.5.1. `buildslist`¶

The `buildslist` command lists all builds that Toaster has recorded. Access the command as follows:

```
$ bitbake/lib/toaster/manage.py buildslist
```

The command returns a list, which includes numeric identifications, of the builds that Toaster has recorded in the current database.

You need to run the `buildslist` command first to identify existing builds in the database before using the `builddelete` command. Here is an example that assumes default repository and build directory names:

```
$ cd ~/poky/build
$ python ../bitbake/lib/toaster/manage.py buildslist
```

If your Toaster database had only one build, the above `buildslist` command would return something like the following:

```
     1: qemux86 poky core-image-minimal
```

4.5.2. `builddelete`¶

The `builddelete` command deletes data associated with a build. Access the command as follows:

```
     $ bitbake/lib/toaster/manage.py builddelete build_id
```

The command deletes all the build data for the specified *`build_id`*. This command is useful for removing old and unused data from the database.

Prior to running the `builddelete` command, you need to get the ID associated with builds by using the `buildslist` command.

4.5.3. `perf`¶

The `perf` command measures Toaster performance. Access the command as follows:

```
     $ bitbake/lib/toaster/manage.py perf
```

The command is a sanity check that returns page loading times in order to identify performance problems.

4.5.4. `checksettings`¶

The `checksettings` command verifies existing Toaster settings. Access the command as follows:

```
     $ bitbake/lib/toaster/manage.py checksettings
```

Toaster uses settings that are based on the database to configure the building tasks. The `checksettings` command verifies that the database settings are valid in the sense that they have the minimal information needed to start a build.

In order for the `checksettings` command to work, the database must be correctly set up and not have existing data. To be sure the database is ready, you can run the following:

```
     $ bitbake/lib/toaster/manage.py syncdb
     $ bitbake/lib/toaster/manage.py migrate orm
     $ bitbake/lib/toaster/manage.py migrate bldcontrol
```

After running these commands, you can run the `checksettings` command.

4.5.5. `runbuilds`¶

The `runbuilds` command launches scheduled builds. Access the command as follows:

```
     $ bitbake/lib/toaster/manage.py runbuilds
```

The `runbuilds` command checks if scheduled builds exist in the database and then launches them per schedule. The command returns after the builds start but before they complete. The Toaster Logging Interface records and updates the database when the builds complete.